Title

Beyond Meat Grinders: An Analysis Framework Addressing the Scale and Complexity of Large Data Sets

Authors

Childs, Hank
Miller, Mark

Publication Date

2006

Peer reviewed

Beyond Meat Grinders:
An Analysis Framework Addressing the Scale *and Complexity* of Large Data Sets

Hank Childs, Mark Miller
Lawrence Livermore National Laboratory
7000 East Avenue, Livermore, Ca, 94550
{childs3 | miller86}@llnl.gov

Keywords: *large data, visualization, data flow networks, contract-based system, subset inclusion lattice*

Abstract

R. W. Hamming said "the purpose of computing is insight, not numbers." We offer a variation: "the purpose of visualization is insight, not pretty pictures." Simply applying standard visualization techniques to large scale data sets all too often results in pretty pictures that fail to convey true insight. Much previous work in the large data area has focused on handling the scale of the data. This work is necessary, but not sufficient. Conclusive analysis of large scale data involves two key ingredients: 1) an architecture to handle the scale of the data; 2) features to manage the complexity of the data. In this paper, we present a system that can not only process large data sets, but also reduce the complexity of the data to produce visual and quantitative results that provide true data insight.

1 INTRODUCTION

As supercomputers become larger and larger, simulations are producing more and more data. A single simulation can now operate on billions of elements and produce terabytes of data. The role of post-processing tools is to analyze the results of these simulations using visual and quantitative techniques. These tools must not only be able to handle the scale of the data, but also perform operations that reduce the complexity of the results. For example, the tool must be able to perform standard visualization operations on large data sets, such as volume rendering or calculating an isosurface. Unfortunately, the resulting images may be very complex. In addition, too many analysis tools are designed as "meat grinders" – taking in complex data and producing equally complex pictures, with little to no effort made to present the data in a way to facilitate the user's deeper understanding. To counteract this trend, tools need to provide general capabilities to help users manage complexity. Examples include reducing the amount of data studied (for example to a subvolume), or employing quantitative techniques, such as calculating the surface area of an isosurface.

The framework described in this paper has been implemented in VisIt [1]. VisIt is a richly featured visualization and analysis tool. Its focus is on large data and unusual data models. The tool has been used on many large data sets, including a recent twenty-seven billion element simulation. One of VisIt's key strengths is that all of its features are interoperable, allowing end users to combine features such as derived quantity generation, data manipulation, plotting variations, and quantitative techniques all in the same analysis.

We strongly believe that richly featured tools with interoperable feature sets are superior to specialized applications when it comes to performing conclusive data analysis. However, by providing a wide range of features, there is a key drawback. Many specialized applications are designed around optimizing the workflow for a handful of operations. For example, a tool that does only volume rendering may build its I/O, data management, communication, and rendering algorithms around a specific volume rendering workflow. With a richly featured tool, this is not practical because of the potential interactions between the operations. A richly featured tool must incorporate general strategies for I/O, data management, communication, and rendering that support all its operations.

An overview of VisIt's framework will be given in the next section, **Handling Large Scale Data**. After describing the framework, we describe some of the techniques that enable a user to gain true insight in the following section, **Reducing Complexity of Large Scale Data Visualizations**.

2 HANDLING LARGE SCALE DATA

VisIt is based upon a client/server design where the server is parallelized. The motivations for this design are similar to those described in [2] – processing data in parallel on the machine it was generated on, while achieving interactive frame rates. It should be noted, however, VisIt's rendering paradigm, outlined in subsection 2.3, is different from that described in [2].

This is LLNL Report UCRL-CONF-217034.

The general data management strategy for the parallel server can be characterized as a variation on a "Scatter-Gather" algorithm, which we call *Scattered-Gather*. The process can be characterized in three steps:

- I/O (scattered)
- Data Flow Networks (processing)
- Rendering (gather)

There is a key distinction between Scatter-Gather and Scattered-Gather. In a traditional Scatter-Gather algorithm, the Scatter phase involves partitioning and distributing data across processors using parallel communication. In a Scattered-Gather algorithm, the data is effectively pre-partitioned (by the simulation) before the post-processing tool reads it. With Scattered-Gather, the data can be read and distributed in chunks without any communication between processors during the Scattered phase.

2.1 I/O

Each processor of VisIt's parallel server reads a portion of the input data set. This is the mechanism that allows VisIt to "scatter" the data set across its server. The key question during the I/O phase is how to assign portions of the input data set to the processors of the server. Before this question can be answered, first consider how this data is inputted to VisIt.

VisIt can obtain data from a simulation in one of two fundamentally different ways. The first, most common use case is where VisIt runs as a standalone application and the data is read from secondary storage. That is, the data is read from files in a filesystem. The second use case is where VisIt runs as a library linked into a simulation code and data is read directly from primary memory. That is, the data is read directly from the memory of a running simulation. Note that the data may need to be re-formatted to match VisIt's input requirements leading to additional memory overhead. Nonetheless, the memory overhead is well worth it given the potential performance gains of bypassing the filesystem.

Next, when the server processes a portion of the data set, the input data set must be partitioned and distributed across the server's processors. The question becomes: what restrictions does the input data's layout impose on how VisIt can partition and distribute that data for parallel processing? There are three scenarios:

- The data is being read from file(s) and the underlying I/O infrastructure[2] only supports a partitioning scheme fixed by the simulation when the file(s) were created.
- The data is being read from file(s) and the underlying I/O infrastructure supports re-partitioning during read.
- The data is being read from the memory of a running simulation, so the partitioning scheme is fixed by how it is stored in the memory of the simulation.

In a nutshell, if the partitioning of the data is fixed, whether it is because of the I/O library or because it comes directly from the simulation, then VisIt simply accepts that partitioning.

In all cases, the simulation has already partitioned the data at least one time; this partitioning is a one-to-one correspondence between each processor of the parallel simulation code and the elements that that processor operates on. The grouping of elements that belong to a single processor is typically called a *domain*. Examples of I/O libraries that support a partitioning scheme fixed by the simulation at the time the files are created are in [3] and [4]. While it is conceivable to alter this partitioning by operating on sub-portions of domains, there is nothing to be gained unless the underlying I/O library supports partial I/O ([3] and [4] do not). On the other hand, some formats permit re-partitioning of the data when it is read. Examples are described in [5] and [6]. In addition, VisIt has an extensible plugin reader interface that allows for new readers for both of these forms and for common I/O libraries (HDF5, etc).

We will refer to a *chunk* as a group of elements that VisIt processes all at one time. Note that if the partitioning of the data is fixed based on domain decomposition, then chunks correspond directly to domains. Furthermore, in this case, there will likely be more chunks than there are processors on VisIt's server. So VisIt must support *chunk overloading*, where each processor of VisIt's server operates on multiple chunks.

Regardless of the form of the input, VisIt's data readers must be able identify chunks and read those chunks for further processing. When the input data is read from the memory of a simulation, the chunk is simply the elements that the simulation has on that processor. When the data is read from a file with a fixed partitioning, we do chunk overloading with the existing domains defined by the file. And when the input data is read from a file that supports repartitioning, the reader plugin can be made smart enough to dynamically decompose the data so that each processor has an approximately equal-size chunk.

Note that, for each of these cases, visualization operations may introduce artifacts along the boundaries of chunks. These artifacts can, however, be dealt with by adding ghost data. [7] contains a complete description of types of ghost data and the conditions under which they are required.

2.2 Data Flow Networks

Our goal was to build a system with many interoperable features, so it was an obvious choice to use data flow networks [8] [9] [10]. VisIt has its own implementation of data flow networks, which was necessary to incorporate *contracts* [7]. It should be noted that many of the modules in VisIt's data flow network implementation offload common visualization operations to a third party library (VTK [8]). Contracts allow VisIt to adaptively employ optimizations specific to the operations being performed in a data flow network. This in turn allows VisIt to handle the scale of massive data sets. Examples of optimizations employed by VisIt are:

[2] The I/O library and/or application code that interfaces with it.

- eliminating chunks from processing that do not affect the final picture
- assigning chunks to processors in an optimal way while not negatively impacting parallel communication
- identifying and creating only the minimum necessary form of ghost data

When the client asks the server to perform a calculation, each processor of the server sets up an identical data flow network. They only differ in the list of chunk(s) from the data set that they process.

VisIt's data flow networks can process chunks using either in-core or out-of-core techniques. Out-of-core techniques are necessary for data sets so large that they cannot fit in primary memory. When a data flow network executes in an out-of-core mode, then one chunk is processed per network execution. In-core techniques are necessary for algorithms that require collective communication. When a data flow network executes in an in-core mode, then all of the chunks travel down the network together and there is only one execution. Of course, we present no solution for handling data sets that are so large that they can only be processed out-of-core in conjunction with algorithms that require the entire data set to be in-core. This is an active research issue.

2.3 Rendering

VisIt uses an adataptive rendering strategy based on the size of a surface to be rendered. If the surface to be rendered contains a small number of geometric primitives, then the server will send that surface to the client to be rendered locally using graphics hardware. However, this approach is inherently non-scalable. So VisIt has an alternative, parallel rendering mode that will accommodate surfaces made up of large numbers of geometric primitives. VisIt has a user-settable heuristic for how large a surface can be before it thinks it will overwhelm the client. Alternatively, users can explicitly choose which rendering mode they would like to use.

Many papers describe techniques for rendering in parallel. See [11] and [12] for examples. The parallel rendering algorithms implemented in VisIt are not new. However, they are versatile and integrate seamlessly with VisIt's client/server architecture. So, we describe them here for completeness.

In parallel, each processor's data flow network produces a portion of the data set to be rendered. Each processor then renders its portion of the data independently, either by using graphics cards (if available), or by using Mesa [13], a software implementation of the OpenGL interface. No attempt is made to re-distribute the data in image space. So, each processor's output image is equal in size to the intended final image and consists of both color and depth values. The individual images are composited in parallel to produce the final image (see figure 1).

If the scene includes semi-transparent surfaces or volumes, then rendering is performed in multiple passes. Opaque geometry, if any, is rendered and composited in the first pass to provide a background image for subsequent passes. During the next pass, data is re-distributed as appropriate for the particular rendering technique (semi-transparent geometry or ray-casted volume rendering) and combined with the background image from the first pass (which includes depth information). If shadows are desired, another rendering pass is made for *shadow maps* [14].

After rendering is completed, the image is compressed and shipped to the client. When multiple servers are used, each server ships color and depth values to the client and a final compositing step is performed there. Otherwise, only color values are shipped to the client.

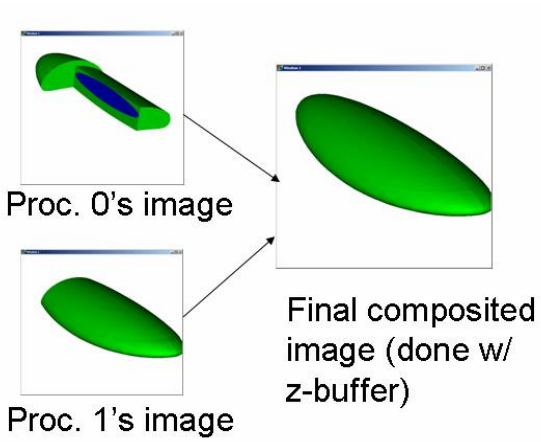

Figure 1: Parallel compositing using color and depth information.

3 REDUCING COMPLEXITY OF LARGE SCALE DATA SETS

Our strategy for reducing data set complexity is to provide functionality in multiple areas. In VisIt, complexity can be reduced in two distinct ways:

- process some of the data; a specific region of interest
- process all of the data set, but simplify it somehow

Of course, these two methods can be combined as well.

The first method, process some of the data, is important when an end user wants to explore what is happening in a specific region. Examples range from examining an area where two materials collide in an engineering simulation to looking at an area of mixing in a turbulence calculation. The first two sections, **Culling By Reference** and **Culling By Value**, describe techniques that allow the user to cull data and focus on a specific region of interest.

The second method, simplifying the data set, is described in the **Quantitative Analysis** section, which deals with operations that dramatically reduce the scale of the data.

3.1 Culling By Reference

Simulations often decompose their data in a variety of ways: by files, by materials, by parts in an assembly, by processor pieces, by levels and patches in an AMR hierarchy, etc. In addition, there are often key subsets in the data, such as nodes and/or zones representing boundary conditions, slide surfaces, user-defined tracers and probes.

We call the technique for managing complexity described in this subsection *Culling By Reference*. The technique enables the user to select which parts of the data set are culled and which are displayed using pre-defined subsets of their data (like the examples described in the paragraph above). VisIt uses a concept from the Sets and Fields (SAF) library [6] called a Subset Inclusion Lattice (or SIL). The SIL incorporates concepts from graph theory and set theory and enables users, for example, to turn on and off certain materials or processor pieces in the display, or to vary the AMR resolution at which data is being displayed.

The SIL is in an encoding of subset relations in graph form. Strictly speaking, a SIL is a directed, acyclic graph of the inclusion (subset-of) relationships of partially ordered sets representing the join-irreducible infinite point-sets of some computational mesh (see [15]). While VisIt's SIL does not support this strict mathematical interpretation, it is similar in concept and is nonetheless called a SIL, although it might be more appropriate named a *lite-SIL*.

Now let's discuss the form of VisIt's SIL in more detail. VisIt's SIL encodes relations between the sets (such as subset relations), as well as the corresponding *category* of a set (for example the **aluminum** set is a subset of the **whole** set using the **material** *category*). The SIL is a graph made up of two sets of nodes: one set corresponds to categories placed on the data, the other corresponds to subsets of the whole data set. Edges in the graph always are incident to both sets – describing relationships between subsets and categories, making our graph bipartite.

Now consider a more concrete example of this graph (see Figure 2). One node of the graph corresponds to the whole data set and it is referred to as the **whole**. **materials** and **processors**, two category nodes, have directed edges to **whole.** Under **processors**, there are many subset nodes, **Proc *X***, each one corresponding to the subset of the data set from processor X. More subset nodes – **plastic**, **air**, and **aluminum –** have directed edges to **materials**.

The SIL is powerful because it is a general framework for representing diverse data in an intuitive way. It provides ultimate flexibility in creating multiple subset hierarchies of a data set. The SIL may have an arbitrary number of subset and category nodes and may be arbitrarily deep. In addition, VisIt provides users with GUI controls (see Figure 3) to manipulate the SIL. This enables users, for example, to turn on and off familiar subsets of their data that are specific to their simulation, such as certain materials or domain, or to vary the AMR resolution at which data is being displayed. In addition, VisIt's SIL provides valuable controls to the user to throttle the amount of complexity being displayed in any one view.

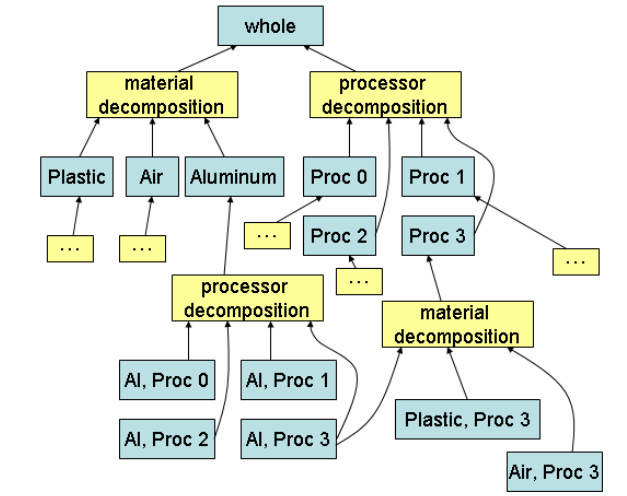

Figure 2: An example graph of the data set. Subset nodes are colored blue, category nodes are colored yellow.

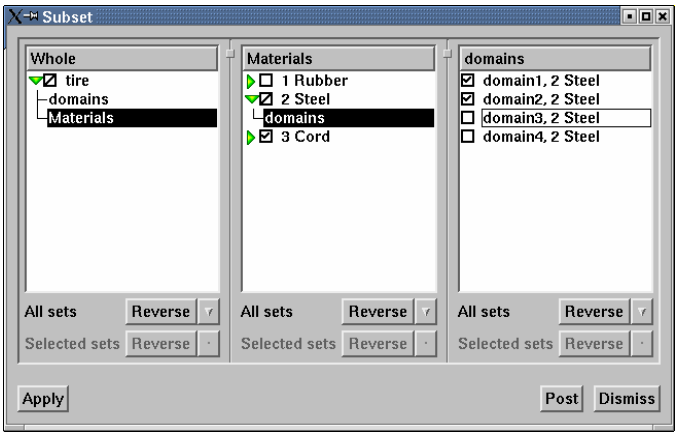

Figure 3: The subset selection window allows users to choose which subsets of the data set they would like to view.

3.2 Culling By Value

In the previous subsection, we described a data culling technique based upon selecting from among pre-defined subsets. In this section, we describe another culling technique based upon data value which we call *Culling By Value*. There are two general techniques for doing this – to eliminate portions based on data values or to eliminate portions based on spatial position. It should be noted that the techniques presented in this section are well understood and are being presented for completeness.

We have found the most powerful operation to cull by value is *isovoluming*. Isovoluming allows a user to specify a range of interest for a specific scalar quantity and then remove all portions of the data set outside that range (see figure 4). For example, a user could restrict the data set to temperatures between forty and fifty degrees Celsius. Of course, some elements may have a portion inside the range and a portion outside the range. In this case, the isovolume

operation will subdivide the element (into tetrahedra) so that the remaining portion is entirely within the range. In addition, the isovolume operation can be applied multiple times with different variables, providing users the ability to ask questions like: "what portion of the data set has pressure between P_1 and P_2 and density between D_1 and D_2?" [16] contains more motivation for these types of sub-selections.

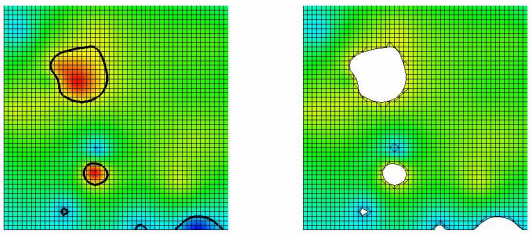

Figure 4: On the left, a data set is shown with thick black lines indicating the areas outside the isovolume's range. On the right, the data set resulting from the isovolume operation is shown.

The ability to make new derived quantities enhances the utility of isovoluming. Derived quantities can transform non-scalar data into scalar data, for example the magnitude of a vector or the maximum shear component of a tensor, which can then be used with the isovolume operation. In addition, the derived quantities can be used to construct new quantities that give greater insight into the data set. A good example comes in [17], which outlines the λ_2-method. This method creates a new derived quantity which identifies vortical flows. The method involves taking the gradient of the velocity field, separating out and then re-combining the symmetric and anti-symmetric portions of the resulting tensors, and solving for its eigenvalues. Vortices can then be identified as connected regions where two of the eigenvalues are negative. These regions can be easily found using the isovolume operation.

When culling by value is applied to coordinate fields, the effect is to cull portions of the data set by spatial position. Many of these techniques are well known – slicing by a plane, slicing by a parameterized surface, clipping by a plane to produce a subvolume, restricting the data set to a box, etc. These techniques are simply enumerated here for completeness. However, the next section, using quantitative techniques, will leverage these operations, as well as the isovoluming technique described above.

3.3 Quantitative Analysis

Within VisIt, the general class of operations returning quantitative information about a data set is called *Queries*. There are two primary classes of queries. The first class, called *Point Queries*, returns information about a single element. Examples of this include the ability to query the variable values of a specific element, or to perform this operation for all time states. The second, more powerful class, called *Aggregate Queries*, calculates information about an entire volume or surface. Examples of this include the ability to calculate centroids or moments of inertia of a volume, integrate a quantity over a volume, calculate the flux along a surface, find the minimum and maximum values of a quantity, etc. Of course, the exact functionality within the direct and aggregate queries required to perform meaningful analysis is dependent on the type of simulation being studied.

One of the most compelling aspects of VisIt's query infrastructure is the way it can be combined with the rest of its feature set. For example, an end user can use the SIL mechanism to limit the data set processed to a single material of interest. The user can then remove all portions of that material on one side of a plane. Then the user can create a new derived quantity that multiplies each element's density with its volume, to give the mass per element. Then an aggregate query can be applied to give the total mass in the volume on the desired side of the plane limited to that material. And this query can be computed as a function of time. This, of course, is just one example of the way that VisIt's features can be combined. But this example motivates the utility of a richly featured application that has strong quantitative capabilities.

In the example above, we focused on specific data manipulations (isovoluming and clipping), derived quantity calculations (mass), and queries (integration). But it is often necessary to add new capabilities in any of these areas. So it is important to keep in mind that VisIt is a highly extensible and customizable framework and that this framework allows for new quantitative features, new data manipulations, and new derived quantities to be added and combined seamlessly. This extensibility is crucial for providing custom analysis for different types of simulations.

More generally, consider how queries fit into data flow networks. Data flow networks typically consist of a source (i.e. file reader), filters, and a sink (i.e. rendering module). But, with queries, we are introducing a new genre of sinks that produce numbers instead of pictures. More importantly, queries are extremely effective at reducing complexity. Also, these queries can be combined with the assets of visualization data flow network implementations – file readers and data manipulation – with all of the associated interoperability.

Finally, VisIt's entire infrastructure is accessible through a command line interface based on Python [18]. This is especially important because it allows users to put together automated "report cards" that describe their data sets. For example, a user could combine the λ_2-method described in **Section 3.2** with the quantitative techniques in this section to determine the amount (i.e. volume) of a simulation undergoing vortical flow. This one number characterizes the data set (and reduces its complexity), allowing it to be compared to other simulations. So these report cards allow for comparison of related simulations. Further, they can be used to compare the thousands of related simulations that occur when doing parameter studies.

4 SUMMARY

A tool that analyzes large data sets must be able to not only handle the scale of that data, but also be able to reduce the complexity of the data being studied. Much work has been devoted to handling the scale of large data sets. But the images produced by applying standard visualization techniques to large, complicated simulations frequently produce only pretty pictures, *not* data insight. Tools that focus solely on handling the scale of the data are just meat grinders – they take complicated data inputs and make complicated picture outputs. Not enough work, in our opinion, has been devoted to managing the complexity of large data sets from within an analysis tool. Restated, handling the scale of the data is a necessary, but not sufficient, condition for large data set analysis. We believe this observation, in and of itself, deserves recognition. In addition, this paper presented a blueprint for addressing the large data problem – both in scale and in complexity.

For completeness, we described our architecture for handling the scale of large data. Although our high-level data flow model is similar to other implementations, for example [19], we believe the I/O model we have presented is the most complete model ever implemented in a richly featured analysis tool.

We also described our techniques for reducing the complexity of a data set. The incorporation of subset inclusion lattices is a novel way to allow end users to remove portions of their data set based on pre-defined sets appropriate for their simulation. And the section on quantitative analysis describes a framework for providing the sort of diverse analysis techniques that are necessary for studying complex data sets. In addition, this framework can be applied repeatedly to the analysis of thousands of related runs, and can make quantitative assessments regarding the differences between two simulations. Finally, the Culling by Value section provided a new variation on some familiar techniques; the techniques described in that section can be used not only to remove complexity by creating a smaller region to explore, but also be part of a toolbox for quantitative techniques.

5 ACKNOWLEDGEMENTS

We thank Wes Bethel from Lawrence Berkeley National Laboratory, who introduced us to the term “meat grinder.” VisIt is developed by B-Division of Lawrence Livermore National Laboratory and the Advanced Simulation and Computing Program (ASC). This work was performed under the auspices of the U.S Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. Lawrence Livermore National Laboratory, P.O. Box 808, L-159, Livermore, Ca, 94551

REFERENCES

[1] http://www.llnl.gov/visit, official website of VisIt.

[2] W. Bethel, 2000. "Visapult - A Prototype Remote and Distributed Application and Framework." Siggraph 2000 Applications and Sketches.

[3] Roberts, L. “Silo User’s Guide”. University of California Research Lab Report, Lawrence Livermore National Laboratory, UCRL-MA-118751-REV-1, 2000.

[4] Schoof, L, Yarberry, V, “EXODUS II: A Finite Element Data Model.” Tech Rep. SAND92-2137, Sandia National Laboratories, 1994.

[5] Pascucci, V.; Frank, R. “Global static indexing for real-time exploration of very large regular grids.” Proc of ACM/IEEE conference on Supercomputing, 2001.

[6] Miller, M, et al. “Enabling Interoperation of High Performance, Scientific Computing Applications: Modeling Scientific Data With the Sets & Fields (SAF) Modeling System.” ICCS-01, May 2001, part II, p158-68.

[7] Childs, H, et al.. “A Contract Based System for Large Data Visualization.” Proc. IEEE Visualization 2005.

[8] Schroeder, W.; Martin, K.; Lorensen, W., 1996. “The design and implementation of an object-oriented toolkit for 3d graphics and visualization.” Proc. Of the 7th conference on Visualization, 1996. IEEE Computer Society Press, 1996, p.93-ff.

[9] Abram, G.; Treinish, L, 1995. “An extended data-flow architecture for data analysis and visualization.” Research report RC 20001 (88338), IBM T. J. Watson Research Center.

[10] Upson, C, et al 1989. “The application visualization system: A computational environment for scientific visualization.” Computer Graphics and Applications, 9(4):30-42, July 1989.

[11] Moreland, K.; Wylie, B.; Pavlakos, C. “Sort-last parallel rendering for viewing extremely large data sets on tile displays.” Proc. IEEE 2001 Symposium on Parallel and Large-Data Visualization and Graphics.

[12] Molnar, S; Cox, M; Ellsworth, D; Fuchs, H, 1994. “A sorting classification of parallel rendering.” IEEE Computer Graphics Applications, 14(4):23-32, 1994.

[13] Paul, B, 1996. “The Mesa 3-D graphics library.” http://www.mesa3d.org

[14] Williams, L, 1978. “Casting Curved Shadows on Curved Surfaces.” Proc. SIGGRAPH, p.270-4, 1978.

[15] Butler, D, Pendley, M. “A Visualization Model Based on the Mathematics of Fiber Bundles.” Computers in Physics, V3 N5, pp. 45-51, 1989.

[16] Stockinger, K, Shalf, J, Wu, K, Bethel, W. “Query-Driven Visualization of Large Data Sets” Proc IEEE Visualization 2005.

[17] Jeong, J, Hussein, F. “On the Identification of a Vortex.” Journal of Fluid Mechanics, p. 69-94, 1995.

[18] Van Rossum, G.; Drake, F. 2003. *The Python Language Reference Manual*. Network Theory Ltd.

[19] Ahrens, J., et al. “Large-scale data visualization using parallel data streaming.” IEEE Computer Graphics Application 21(4):34-41, 2001.